(19)日本国特許庁（ＪＰ）　(12)公開特許公報（Ａ）　(11)特許出願公開番号
特開2002－230065
（P2002－230065A）
(43)公開日　平成14年８月16日(2002.8.16)

(51)Int.Cl.'		識別記号	FI		テーマコード'（参考）
Ｇ０６Ｆ	17/50	６５６	Ｇ０６Ｆ	17/50	６５６Ａ　５Ｂ０４６
		６５４			６５４Ｍ　５Ｂ０４８
		６６４			６６４Ａ
					６６４Ｋ
	11/22	３３０		11/22	３３０Ｂ

審査請求　未請求　請求項の数16　ＯＬ　（全 23 頁）　最終頁に続く

(21)出願番号　特願2001－27432(P2001－27432)

(22)出願日　平成13年２月２日(2001.2.2)

(71)出願人　000003078
株式会社東芝
東京都港区芝浦一丁目１番１号

(72)発明者　松井　伸郎
神奈川県川崎市幸区小向東芝町１番地　株式会社東芝マイクロエレクトロニクスセンター内

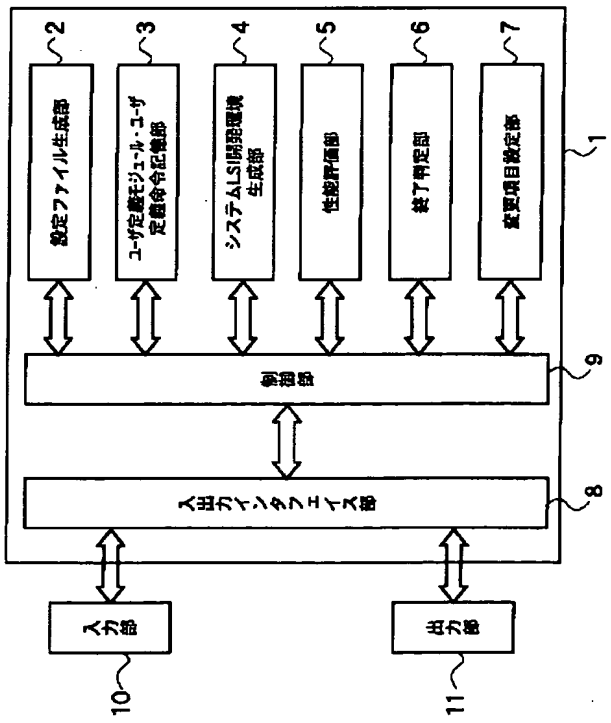
(72)発明者　水野　淳
神奈川県川崎市幸区小向東芝町１番地　株式会社東芝マイクロエレクトロニクスセンター内

(74)代理人　100083806
弁理士　三好　秀和　（外７名）

最終頁に続く

(54)【発明の名称】　システムＬＳＩ開発装置およびシステムＬＳＩ開発方法

(57)【要約】
【課題】　最適な性能のＬＳＩを開発設計することを可能にする。
【解決手段】　オプション命令が定義されたプロセッサを含むシステムＬＳＩの開発設計に利用されるコンピュータシステム上で動作する複数のソフトウェアから成るシステムＬＳＩ開発装置であって、システムＬＳＩの開発設計に関わる可変項目定義情報に基づいて前記ソフトウェアを動作させ、システムＬＳＩのハードウェア記述、検証環境および開発設計ツールを生成するシステムＬＳＩ開発環境生成部を具備し、可変項目定義情報は、オプション命令情報、ユーザ定義モジュールおよびマルチプロセッサ構成に関する情報の少なくとも１つを含む。



【特許請求の範囲】

【請求項１】　オプション命令が定義されたプロセッサを含むシステムＬＳＩの開発設計に利用されるコンピュータシステム上で動作する複数のソフトウェアから成るシステムＬＳＩ開発装置であって、
システムＬＳＩの開発設計に関わる可変項目定義情報に基づいて前記ソフトウェアを動作させ、システムＬＳＩのハードウェア記述、検証環境および開発設計ツールを生成するシステムＬＳＩ開発環境生成部を具備し、
前記可変項目定義情報は、オプション命令情報、ユーザ定義モジュールおよびマルチプロセッサ構成に関する情報の少なくとも１つを含むことを特徴とするシステムＬＳＩ開発装置。

【請求項２】　ユーザ定義モジュールが指定された際には、ユーザ定義モジュールの仕様に対応するハードウェア記述と前記プロセッサとを接続することにより、前記システムＬＳＩのハードウェア記述を生成することを特徴とする請求項１に記載のシステムＬＳＩ開発装置。

【請求項３】　前記ユーザ定義モジュールの仕様に対応するハードウェア記述は、ユーザ定義モジュールの仕様に対応する動作記述を高位合成することにより生成することを特徴とする請求項２に記載のシステムＬＳＩ開発装置。

【請求項４】　マルチプロセッサ構成が指定された際には、プロセッサ毎にハードウウェア記述を生成し、当該ハードウェア記述をバスで接続したハードウェア記述を生成することを特徴とする請求項１～請求項３いずれか１項に記載のシステムＬＳＩ開発装置。

【請求項５】　前記設定ファイル生成部内に格納された可変項目定義情報の可能な全ての組み合わせについて、アプリケーションの性能を並列に評価する性能評価部を具備することを特徴とする請求項１～４いずれか１項に記載のシステムＬＳＩ開発装置。

【請求項６】　前記性能評価部は、アプリケーションのコードサイズ、実行命令数、実行サイクル数、チップのゲートサイズおよび消費電力の少なくとも１つを評価することを特徴とする請求項５に記載のシステムＬＳＩ開発装置。

【請求項７】　前記性能評価部による性能評価結果に基づいて、前記可変項目定義情報を変更する変更項目設定部を具備することを特徴とする請求項１～請求項６いずれか１項に記載のシステムＬＳＩ開発装置。

【請求項８】　前記可変項目定義情報には、ユーザ定義の命令に関する情報、デバイスのオン／オフ情報、デバイスのアドレス情報、デバイスのサイズ情報およびキャッシュのサイズ情報に関する情報の少なくとも１つが含まれることを特徴とする請求項１～請求項７いずれか１項に記載のシステムＬＳＩ開発装置。

【請求項９】　オプション命令が定義されたプロセッサを含むコンピュータシステム上で動作する複数のソフトウェアを利用してシステムＬＳＩを開発設計するシステムＬＳＩ開発方法であって、
システムＬＳＩの開発設計に関わる可変項目定義情報を入力するステップと、
入力された可変項目定義情報に基づいて前記ソフトウェアを動作させ、システムＬＳＩのハードウェア、検証環境および開発設計ツールを生成するステップとを有し、
前記可変項目定義情報は、オプション命令情報、ユーザ定義モジュールおよびマルチプロセッサ構成に関する情報の少なくとも１つを含むことを特徴とするシステムＬＳＩ開発方法。

【請求項１０】　ユーザ定義モジュールが指定された際には、ユーザ定義モジュールの仕様に対応するハードウェア記述と前記プロセッサとを接続することにより、前記システムＬＳＩのハードウェア記述を生成することを特徴とする請求項９に記載のシステムＬＳＩ開発方法。

【請求項１１】　前記ユーザ定義モジュールの仕様に対応するハードウェア記述は、ユーザ定義モジュールの仕様に対応する動作記述を高位合成することにより生成することを特徴とする請求項１０に記載のシステムＬＳＩ開発方法。

【請求項１２】　マルチプロセッサ構成が指定された際には、プロセッサ毎にハードウウェア記述を生成し、当該ハードウェア記述をバスで接続したハードウェア記述を生成することを特徴とする請求項９～請求項１１いずれか１項に記載のシステムＬＳＩ開発方法。

【請求項１３】　入力された可変項目定義情報の可能な全ての組み合わせについて、アプリケーションの性能を並列に評価するステップを有することを特徴とする請求項９～請求項１２いずれか１項に記載のシステムＬＳＩ開発方法。

【請求項１４】　前記評価処理において、アプリケーションのコードサイズ、実行命令数、実行サイクル数、チップのゲートサイズおよび消費電力の少なくとも１つを評価することを特徴とする請求項１３に記載のシステムＬＳＩ開発方法。

【請求項１５】　前記性能評価結果に基づいて、前記可変項目定義情報を変更するステップを有することを特徴とする請求項９～請求項１４いずれか１項に記載のシステムＬＳＩ開発方法。

【請求項１６】　前記可変項目定義情報には、ユーザ定義の命令に関する情報、デバイスのオン／オフ情報、デバイスのアドレス情報、デバイスのサイズ情報およびキャッシュのサイズ情報に関する情報の少なくとも１つが含まれることを特徴とする請求項９～請求項１５いずれか１項に記載のシステムＬＳＩ開発方法。

【発明の詳細な説明】

【０００１】

【発明の属する技術分野】本発明は、少なくともオプション命令が定義されたcofigurableプロセッサを含むシ

ステムＬＳＩの開発設計に利用される、コンピュータシステム上で動作する複数のソフトウェアから成るシステムＬＳＩ開発装置およびそのシステムＬＳＩ開発方法に関する。

【０００２】

【従来の技術】近年、マルチメディア処理等の組み込み機器用ＬＳＩに対するニーズの多様化とその市場サイクルの短期化から、アプリケーションに最適なＬＳＩを短期に開発するＬＳＩ設計開発システムが望まれている。

【０００３】一般に、ＬＳＩ内に汎用プロセッサを搭載した場合、ハードウェアの設計コストはほとんどゼロに等しいが、アプリケーションの性能をフルに引き出すことはできない。このため、最近では、命令やメモリ構成等が選択可能なcofigurableプロセッサが提供されるようになり、また同時に、cofigurableプロセッサの提供者は、コンフィグレーションを指定して論理合成可能なハードウェア記述を出力するシステムを提供している。このようなプロセッサおよびシステムによれば、オプション命令やメモリサイズを指定することによって、アプリケーションに最適な構成のプロセッサを短期間で入手することが可能となる。

【０００４】一方、通常、命令セット等が変われば、コンパイラやシミュレータ等のソフトウェア開発ツールも変更しなければならない。このため、コンフィグレーションを指定することにより、ハードウェア記述と同時に、ソフトウェア開発ツールを生成するシステムも提供されるようになっている。このようなシステムによれば、ソフトウェア開発ツールの設計に要する労力および時間を大幅に削減することができる。

【０００５】

【発明が解決しようとする課題】以上述べたように、configurableプロセッサによれば、アプリケーションに最適なプロセッサ構成を短期間で入手することができると同時に、ソフトウェア開発も従来通りのフローで行うことが可能となる。しかしながら、アプリケーションの性能をより十分に引き出すためには、アプリケーションの一部をハードウェア化して、プロセッサ＋専用ハードウェアというシステムＬＳＩ構成をとることも考えられ、そして、この方が最適なＬＳＩ構成になることが多い。ところが、従来の技術では、カスタマイズすることができるのはプロセッサの構成だけであって、プロセッサの周辺に追加するユーザ定義のハードウェアモジュールを含めた構成を処理することができない。このため、現在、プロセッサ＋専用ハードウェア構成を含めたシステムＬＳＩ構成の検討、性能評価、設計検証およびソフトウェア開発を行うことができず、より最適な性能のＬＳＩを開発することが困難な状態となっている。

【０００６】本発明は、上記の技術的課題を解決すべくなされたものであり、その目的は、最適な性能のＬＳＩを開発設計することを可能にする技術を提供することにある。

【０００７】

【課題を解決するための手段】本発明に係るシステムＬＳＩ開発装置の特徴は、オプション命令が定義されたプロセッサを含むシステムＬＳＩの開発設計に利用されるコンピュータシステム上で動作する複数のソフトウェアから成るシステムＬＳＩ開発装置であって、システムＬＳＩの開発設計に関わる可変項目定義情報に基づいてソフトウェアを動作させ、システムＬＳＩのハードウェア記述、検証環境および開発設計ツールを生成するシステムＬＳＩ開発環境生成部とを具備し、可変項目定義情報は、オプション命令情報、ユーザ定義モジュールおよびマルチプロセッサ構成に関する情報の少なくとも１つを含むことにある。このような構成のシステムＬＳＩ開発装置によれば、オプション命令、ユーザ定義モジュールおよびマルチプロセッサ構成の少なくとも１つを含む可変項目定義情報の可能な全ての組み合わせについて、オプション命令が定義されたプロセッサを含むシステムＬＳＩのハードウェア、検証環境、開発ツールを一貫して網羅的／複数並列に作成することができるので、アプリケーションに最適なシステムＬＳＩを短期間に開発することが可能となる。

【０００８】

【発明の実施の形態】以下、図１～図１９を参照して、本発明の実施の形態のシステムＬＳＩ開発装置の構成およびその動作について説明する。

【０００９】《システムＬＳＩ開発装置の構成》始めに、図１を参照して、本発明の実施の形態のシステムＬＳＩ開発装置の構成について簡単に説明する。

【００１０】本発明の実施の形態のシステムＬＳＩ開発装置１は、図１に示すように、設定ファイル生成部２、ユーザ定義モジュール・ユーザ定義命令記憶部３、システムＬＳＩ開発環境生成部４、性能評価部５、終了判定部６、変更項目設定部７、入出力インタフェイス部８および制御部９を有する。

【００１１】また、本発明の実施の形態のシステムＬＳＩ開発装置１は、装置１内に各種情報を入力するための入力部１０と装置１からの各種情報を出力するための出力部１１とに接続されている。ここで、入力部１０の形態としては、例えばキーボード、マウスポインタ、テンキー、タッチパネル等を採用することができ、また、出力部１１の形態としては、例えばディスプレイ装置や印刷装置等を採用することができる。

【００１２】次に、図２～図１３を参照して、このシステムＬＳＩ開発装置内の各部の構成について順に詳しく説明する。

【００１３】『設定ファイル生成部の構成』設定ファイル生成部２は、図２に示すように、入力解析部２１、ローカルメモリマップ生成部２２、オプション情報記憶部２３、デバイス構成記憶部２４、キャッシュ構成記憶部

２５およびローカルメモリマップ記憶部２６を備え、システムＬＳＩの設計上のコンフィグレーションを記述したコンフィグレーション指定ファイル、可変値設定情報、最優先項目設定情報、目標性能指定情報（ハードウェア性能、ソフトウェア性能の一方又は両方を含み、ハードウェアの性能指標としては面積、周波数、消費電力、ソフトウェアの性能指標としては、コードサイズ、実効命令数、実行サイクル数を含む。）および変更項目設定部７からの変更項目リスト情報に基づいて、システムＬＳＩ設計上のコンフィグレーションを生成、格納する。

【００１４】以下、設定ファイル生成部２内のそれぞれの構成要素について説明する。

【００１５】〈入力解析部の構成〉入力解析部２１は、コンフィグレーション指定ファイルの内容を解析し、コンフィグレーション指定ファイル内に記述されたコンフィグレーションの内容をオプション情報記憶部２３、デバイス構成記憶部２４およびキャッシュ構成記憶部２５の各部に分割する。

【００１６】ここで、この実施の形態においては、コンフィグレーション指定ファイル内には、オプション命令の有無、デバイスの有無（デバイスには、命令メモリ、データメモリ、コプロセッサが含まれ、命令メモリ、データメモリの可変項目には、サイズ、アドレスが含まれる）、キャッシュの有無、キャッシュサイズ、ユーザ定義命令、ユーザ定義ハードウェアモジュール、ＬＳＩ内部のメモリマップ（メモリマップには、外部メモリ領域、内部メモリ領域、入出力領域が含まれ、外部メモリ領域には、キャッシュアクセス領域の指定項目も含まれるものとする。また、各メモリ領域にはレイテンシ情報も含まれる）、マルチプロセッサ構成（マルチプロセッサの数など）等の情報を記述するものとする。なお、ユーザ定義命令やユーザ定義ハードウェアモジュールの記述部分には、ユーザ定義命令やユーザ定義モジュールを記述したファイル名の記憶位置（この実施の形態においては、ユーザ定義モジュール・ユーザ定義命令記憶部３の位置）を絶対パスや相対パス等の記述によって指定するようにする。

【００１７】また、コンフィグレーション指定ファイルからシステムＬＳＩ設計上のコンフィグレーションを指定する際は、ユーザは、設定ファイル生成部２内の各記憶部２３～２６で定義されている可変値の中から１つの値を選択し、選択した値を、例えば図３に示すようなコンフィグレーション指定ファイル内に記述する。

【００１８】すなわち、例えば、以下に示すＳＩＺＥ記述においては、内蔵データメモリサイズが１，２，４，８，１６，３２[ＫＢ]のいずれかが選択可能であり、このＳＩＺＥ記述を省略した場合には、８[ＫＢ]になることを示す。

【００１９】DMEM:

SIZE Ptype={1, 2, 4, 8, 16, 32}, default=8, comment="内蔵データメモリのサイズ";

また、以下に示すＩＳＡ＿ＤＥＦＩＮＥ記述においては、ユーザ定義命令のＩＳＡ定義ファイル名として任意の文字列を記述しなければならないことを示す。

【００２０】UCI:

ISA_DEFINE Ptype=string, default="", comment="ユーザ定義命令のＩＳＡ定義ファイル名";

なお、上記のコンフィグレーションの設定作業は、入出力インタフェイス部８を介した対話的な処理によって行うようにしても良い。コンフィグレーションの設定を対話的に行うようにすることにより、ユーザは、コンフィグレーション指定ファイルの文法等を意識せずに、コンフィグレーションを容易に設定することができ、直接ファイルに記述する場合と比べてコンフィグレーションの設定に要する時間を短縮することができる。さらに、全てのコンフィグレーションの設定を対話的に行えるので、コンフィグレーションの１つ設定し忘れる等の間違いを防止することができる。

【００２１】また、コンフィグレーションに可変値を設定するようにしても良い。コンフィグレーションに可変値を設定した場合には、設定ファイル生成部２は、基本設定、可変値の測定用に作られたテンプレート、ユーザが設定した値のいずれかを用いて自動的に他のコンフィグレーションの設定値を導出するようにする。コンフィグレーションに可変値を設定することにより、ユーザは、コンフィグレーションの可変範囲に応じた複数の開発環境、検証環境を同時に得ることができるので、ＬＳＩ開発サイクルの短縮を実現することが可能となる。

【００２２】さらに、コンフィグレーションの中で最優先項目を設定するようにしても良い。コンフィグレーションの中で最優先項目が設定された場合、設定ファイル生成部２は、最優先項目以外の部分の設定値については、基本設定、最優先項目に基づいてテンプレート又はユーザが指定した設定値を用いて導出し、最優先項目が最適となるようなコンフィグレーションを生成する。コンフィグレーションの中に最優先項目を指定することにより、ユーザはシステムＬＳＩの構成を考えることなく、優先項目に適した構成を開発環境、検証環境を作成し、その評価を行うことができる。

【００２３】〈ローカルメモリマップ生成部の構成〉ローカルメモリマップ生成部２２は、コンフィグレーション指定ファイルから指定されるグローバルマップ（全プロセッサに共通のメモリ領域、例えば、図４（ａ）参照）と、命令メモリ、データメモリ、命令キャッシュ、データキャッシュといったシステムＬＳＩ内の個々のプロセッサのローカルメモリ情報とを統合して、図４（ｂ）に示すようなシステムＬＳＩ内のプロセッサ毎のローカルメモリマップを生成し、生成したローカルメモリマップをローカルメモリマップ記憶部２６内に格納す

る。ここで、プロセッサのローカルメモリ領域は、個々のローカルメモリ毎に予約されているグローバルマップ中のメモリ領域に対して、コンフィグレーション指定ファイルにより指定されるメモリサイズを反映させることによって生成することができる。また、予めグローバルマップ中に各プロセッサのシャドウメモリ情報を指定しておくようにすれば、各プロセッサのローカルマップに他のプロセッサのローカルメモリを含むシャドウメモリを作ることもできる。

【００２４】〈オプション情報記憶部の構成〉オプション情報記憶部２３は、入力解析部２１によるコンフィグレーション指定ファイルの解析結果に基づいて、システムＬＳＩに内蔵されるプロセッサの命令セット内のオプション命令のＯＮ／ＯＦＦ（オン／オフ）の種別に関する情報を格納する。

【００２５】〈デバイス構成記憶部の構成〉デバイス構成記憶部２４は、入力解析部２１によるコンフィグレーション指定ファイルの解析結果に基づいて、システムＬＳＩに内蔵されるデバイスのＯＮ／ＯＦＦの種別に関する情報とそのサイズに関する情報、アドレス情報、プロセッサの命令メモリやデータメモリに関する情報、コプロセッサ等のオプションハードウェアに関する情報を格納する。

【００２６】〈キャッシュ構成記憶部の構成〉キャッシュ構成記憶部２５は、入力解析部２１によるコンフィグレーション指定ファイルの解析結果に基づいて、システムＬＳＩのキャッシュのＯＮ／ＯＦＦの種別、キャッシュのタイプ（direct, 2-way, 4-way, n-way）に関する情報とそのサイズに関する情報を格納する。

【００２７】〈ローカルメモリマップ記憶部の構成〉ローカルメモリマップ記憶部２６は、ローカルメモリマップ生成部２２が生成したローカルメモリマップに関する情報を格納する。

【００２８】『ユーザ定義モジュール・ユーザ定義命令記憶部の構成』ユーザ定義モジュール・ユーザ定義命令記憶部３は、ユーザ定義のハードウェアモジュールおよびシステムＬＳＩに内蔵するプロセッサの命令セットにおけるユーザ定義命令に関する情報を格納する。ここで、ユーザ定義のハードウェアモジュールに関する情報はＲＴＬ記述、又は動作記述、命令の動作に関する情報はＣモデルで記述して記憶部３内に格納することが望ましい。動作記述とＣモデル記述は同一のものでも良い。また、ユーザ定義命令に関する情報は、コンフィグレーション指定ファイルから指定される、図４（ｃ）に示すようなＩＳＡ定義ファイルに記述して格納することが望ましい。

【００２９】『システムＬＳＩ開発環境生成部の構成』システムＬＳＩ開発環境生成部４は、図２に示すように、ＲＴＬ生成部４１、シミュレータカスタマイズ部４２、コンパイラカスタマイズ部４３、アセンブラカスタマイズ部４４および検証ベクトル生成部４５を備え、設定ファイル生成部２内に格納されているコンフィグレーションの可能な全ての組み合わせについて、システムＬＳＩのハードウェア、検証環境および開発設計ツールを生成する。

【００３０】以下、このシステムＬＳＩ開発環境生成部の内部構成について詳しく説明する。

【００３１】〈ＲＴＬ生成部の構成〉ＲＴＬ生成部４１は、設定ファイル生成部２の記憶部内に記憶されているコンフィグレーションに基づいて、システムＬＳＩに内蔵されるプロセッサのＲＴＬ記述を生成する。

【００３２】具体的には、図６に示すように、ブロック接続部４１ｄが、デバイス構成記憶部２４とキャッシュ構成記憶部２５内に格納されたコンフィグレーションを参照してユーザにより設定されたコンフィグレーションに対応するＲＴＬテンプレート４１ａ，４１ｂを選択し、プロセッサコア部のＲＴＬ記述４１ｃに対して、選択したＲＴＬテンプレートと、高位合成処理部４１ｅがユーザ定義モジュール・ユーザ定義命令記憶部８内に格納されたユーザ定義命令又はユーザ定義モジュールの仕様である動作記述を高位合成することにより生成される、ＲＴＬ記述とを接続することにより、プロセッサのＲＴＬ記述を生成する。なお、記憶部８内の仕様がＲＴＬ記述である場合は、そのままインタフェイス信号が合うように接続する。

【００３３】なお、マルチプロセッサ構成が定義された場合には、ブロック接続部４１ｄは、複数のプロセッサ記述を生成し、それらをバスで接続したマルチプロセッサのＲＴＬ記述を生成するものとする。

【００３４】また、ＲＴＬテンプレート４１ａ，ｂ内に含まれる命令メモリ、データメモリ、オプションハードウェア、命令キャッシュおよびデータキャッシュについては、ユーザが選択可能なメモリサイズの各々についてＲＴＬ記述を予め用意し、指定されたコンフィグレーションに応じてＲＴＬ記述を選択接続するようにする。また、オプション命令については、オプション命令に対応するハードウェアのテンプレートをオプション命令のＯＮ／ＯＦＦ全ての組み合わせについて用意する。

【００３５】例えば図７に示す例においては、除算オプション命令（ＤＩＶ）、最大最小値オプション命令（ＭＩＮＭＡＸ）各々のＯＮ／ＯＦＦの組み合わせ４通りについて、命令機能ユニットのＲＴＬ記述をそれぞれテンプレートとして用意し、デバイス構成記憶部２４とキャッシュ構成記憶部２５内に格納されたコンフィグレーションに合わせて選択接続する。また、コプロセッサ等のオプションハードウェアについても同様、コンフィグレーションで有効と指定されている場合、既定義のＲＴＬ記述を接続する。さらに、ユーザ定義のハードウェアがある場合、ユーザが記述したハードウェア記述を接続する。

【００３６】以上の接続処理によって、ＲＴＬ生成部４１は、設定されたコンフィグレーションに対応したプロセッサのＲＴＬ記述を生成する。例えば、プロセッサコアに４[ＫＢ]の命令メモリと８[ＫＢ]のデータメモリを追加したプロセッサを最初に構築する場合、ユーザはコンフィグレーションのパラメータ指定のみを行えば、プロセッサのＲＴＬ記述を自動的に得ることができる。また、命令メモリ、データメモリのサイズを変更し、コプロセッサと命令キャッシュ、データキャッシュを追加し、さらに幾つかのオプション命令とユーザ定義命令を持つプロセッサ２を構築する場合であっても、コンフィグレーションのパラメータ指定を変更し、ユーザ定義命令に対応するハードウェアのＲＴＬ記述を与えるのみで、新たなプロセッサのＲＴＬ記述を得ることができる。また、複数のＲＴＬテンプレートを用意しておくのではなく、可変項目をパラメータ表現したＲＴＬテンプレートにコンフィグレーションで設定された値を反映させることによっても同等の効果が得られる。さらに、このようなパラメータ化されたＲＴＬテンプレートは、メモリ、キャッシュ等の部分モジュール毎でも良く、それらを含んだプロセッサ全体のものでも良い。全てのパラメータを包含した１つのＲＴＬテンプレートでも良い。

【００３７】〈シミュレータカスタマイズ部の構成〉シミュレータカスタマイズ部４２は、コンフィグレーションに沿った命令動作を実行するシミュレータを生成する。

【００３８】具体的には、図８に示すように、再コンパイル部４２ｃが、シミュレータテンプレート４２ａにユーザ定義モジュール・ユーザ定義命令記憶部３内に格納されたユーザ定義ハードウェアのＣモデルを組み込んで再コンパイルすることにより、シミュレータを再構築する。また、起動オプション情報抽出部４２ｄは、設定ファイル生成部内のデバイス構成記憶部２４とキャッシュ構成記憶部２５に格納されているデータを参照して、起動時のオプションを指定するシミュレータ起動オプションファイル（図９（ａ）参照）を生成する。そして、組み合わせ処理部４２ｅは、再構築したシミュレータとシミュレータ起動オプションファイルとを組み合わせることにより、コンフィグレーションに沿った命令動作を実行するシミュレータを生成する。

【００３９】なお、シミュレータは、特定アドレス通過時に、デバッグ指令の実行結果を出力する手段を備えることが望ましい。従来、実行途中の誤りが起きた場合にエラーを出力するアサーションエラープログラム等があったが、正常に実行している時の途中経過を調べるためには、途中結果を出力する命令をアプリケーションに埋め込んでおくか、デバッガ等で実行を停止させて途中経過を読み取るしかなかった。起動時に指定されるアドレスを通過した際に所定の情報を出力するようにすれば、シミュレータでアプリケーションを実行している途中に

途中経過を調べることが可能となる。例えば、次に示すようなコマンドでシミュレータを起動する。

【００４０】sim -load test.hex -printx mem(0x400) at 0x1200

ここで、sim -load test.hex, -printx mem(0x400) at 0x1200はそれぞれ、システムＬＳＩのシミュレータ、test.hexというファイルをロードして実行するという命令、0x1200番地を通過したときにメモリの0x400番地の内容を１６進数で出力する命令を示す。

【００４１】シミュレータは、この引数で指定された指示を記憶しておき、プログラムカウンタが0x1200番地を通過する度に0x400番地の内容を１６進数でコンソールに出力する。このとき、アプリケーションプログラムには観測用のコードが埋め込まれているわけではないので、命令数やサイクル数等の統計情報には全く影響を及ぼさないで観測できることになる。また、特定アドレスで実行を停止させ、人手を介してメモリの値を読み出し表示させる等の作業を行う必要がないので、統計情報を正確に取りながら、特に注目したい途中結果を確認しながらアプリケーションプログラムのシミュレーションをすることができるようになり、効率の良いＬＳＩ開発が可能となる。

【００４２】また、シミュレータは、指定された以外の領域のメモリアクセスが起きた場合に実行を停止することが望ましい。設定されたコンフィグレーションに従ってシミュレータを起動した時には、使用可能なメモリが明確に与えられている。もし、実施の基板上で実装されていない領域のメモリをアクセスするとバスエラーになる等してそれ以後の動作は保証されない。動作が保証されないプログラムは、シミュレータ実行時に間違いを指摘される等して、早期に修正しておく必要がある。そこで、シミュレータは存在するメモリ領域をを指定され、明示的に指定されない領域はアクセス禁止にして、違反があったら警告する（且つ停止する）。また、シミュレータが対話モードを有志、対話モード（デバッグモード）に移行し、どこが原因だったのかが解析できるようになっている。また、シミュレータがデバッガと通信して動作している場合には、シミュレーション動作を中断してデバッガのコマンド待ちに移行し、無効な領域をアクセスしたというエラーを表示し、プログラムの誤りを早期に確実にユーザに伝える。また、メモリのデコード回路では、上位のアドレス信号線の接続を省略されて構成される場合がある。この場合、ターゲットでアドレスを間違えたプログラムを実行してもバスエラーにはならないが、アクセスが書き込みだった場合には、意図しないアドレスの内容を書き換えてしまい、後の実行結果が期待通りにならない時がある。この間違ったプログラムをＲＴＬシミュレータで実行した場合でも、こういった誤りはすぐには発見することができない。しかし、シミュレータに正しいアドレスのみをアクセスできるように

正確なマッピング情報を与えて実行させたときには、指定範囲以外のアドレスをアクセスしたことが即座に警告されれば、ユーザは間違ったプログラムに早期に気付くことになり、無駄な解析をする時間を節約でき、開発サイクルの短縮が図れる。

【００４３】なお、デバッガについては、デバイス構成記憶部２４とキャッシュ構成記憶部２５内に格納されているデータを参照して、図９（ｂ）に示すようなデバッガ起動オプションファイルを生成する。デバッガ起動オプションファイルを利用することにより、デバッガはコンフィグレーションに沿った動作を行うことが可能となる。

【００４４】また、デバッガは、シミュレータの統計情報を仮想的なレジスタ若しくは変数として読み出す手段を有することが望ましい。デバッガは、$profileという変数を特殊な変数として予約してある。統計情報は、領域毎に管理されている。そして、デバッガの変数読み出しコマンドprintによって表示させる。領域の３番目の情報を表示させるためには、print $profile[3]のように入力する。

【００４５】dbg> print $profile[3]
profile3:Count=7 Inst=123 Cycle=145
dbg>

さらに、言語ツールは、シミュレータ用のデバッグ用指令をアドレスと共にシンボルやソース行と同等なデバッグ情報として出力する手段を有することが望ましい（ターゲットの機械命令には反映されない）。コンパイラには、シミュレータ等で標準出力に途中結果等を出力するための拡張が施されている。#pragma cosoleoutに続く文は、ターゲットの機械語には翻訳されずに、シンボル名や行番号と同様、デバッグ情報としてファイルに格納される。シミュレータは、デバッグ情報付きののオブジェクトファイルを読み込んだとき、#pragma文が定義されたアッドレス情報を記憶すると共に、デバッグ出力情報であるprintf("a=%d¥n", a)を記憶する。シミュレータが該当アドレスを実行しようとしたときに、デバッグ情報を出力しなければならないと判断し、printf文をパーサに送りデバッグ情報を出力する。

【００４６】このように、ターゲットの機械命令に影響を与えずに、シミュレータ実行時にデバッグ情報を出力し、シミュレータの実行も中断せずに内容を確認することもできる。このとき、シミュレータ実行時に収集される統計情報は、ターゲットで実行されるものと同一の情報である。別の表現をすれば、ターゲットとシミュレータで同一のオブジェクトコードを用いることができることになる。したがって、実行環境の違いによる再コンパイルなどの時間が発生しなくなる分だけ開発サイクルも短縮される。また、単一のオブジェクトファイルを共有できるので、管理が容易になることは明白である。

【００４７】


```
func(int b){
float a;
for(I=1;I<20; i++){
        A=b/I;
#pragma consoleout printf("a=%d¥n",a);
}
```

〈コンパイラカスタマイズ部の構成〉コンパイラカスタマイズ部４３は、オプション情報記憶部２３とユーザ定義命モジュール・ユーザ定義命令記憶部３内に格納されたデータを参照して、機械命令関数宣言を含む、図９（ｃ）に示すような機械命令関数宣言ヘッダファイルを生成する。なお、ここでいう機械命令関数宣言とは、プロセッサ固有の命令を高級言語から直接指定するために、プロセッサ固有の命令を高級言語の関数宣言として記述したものである。

【００４８】具体的には、図１０に示すように、機械命令関数宣言抽出部４３ｃが、ユーザ定義モジュール・ユーザ定義命令記憶部３内に格納されたユーザ定義命令を参照して、対応する機械命令関数宣言を抽出する。また、結合処理部部４３ｄが、オプション情報記憶部２３内に格納されたデータを参照して、既定義のテンプレート４３ｂから有効なオプション命令に対応する機械命令関数宣言を選択する。そして、結合処理部４３ｄは、テンプレート４３ｂと機械命令関数宣言抽出部４３ｃそれぞれから抽出された機械命令関数宣言を結合することにより、機械命令関数宣言ヘッダファイルが生成する。この機械命令関数宣言ヘッダファイルには、コンフィグレーションで有効になったオプション命令とユーザ定義命令に対応する機械命令関数宣言が含まれる。

【００４９】これにより、ユーザは、高級言語プログラムからコンフィグレーションで指定したオプション命令とユーザ定義命令を直接指定し、利用することが可能となる。

【００５０】なお、コンパイラカスタマイズ部３３はオプション命令抽出部４３ａを有し、オプション命令抽出部４３ａは、オプション情報記憶部２３内に格納されたデータを参照して、最適化に利用することができるオプション命令の情報を取得し、コンパイラ起動時のオプションを指定するコンパイラ起動オプションファイル（図１１（ａ）参照）を生成する。

【００５１】これにより、利用可能なオプション命令の情報をコンパイラの最適化に反映させることができる。また、コンパイラの関数ライブラリは、起動オプションファイルを使用してソースから再コンパイルされるので、利用可能なオプション命令を組み込んだライブラリを生成することができる。

【００５２】〈アセンブラカスタマイズ部の構成〉アセンブラカスタマイズ部４４は、利用可能なオプション命令とユーザ定義命令のニーモニックと命令形式の情報を組み込んで、アセンブラを再構築する。このアセンブラ

カスタマイズ部４４によれば、利用可能な全ての命令の任意の組み合わせからなるアセンブラプログラムについて、対応するオブジェクトコードを生成することができる。

【００５３】〈検証ベクトル生成部の構成〉検証ベクトル生成部４５は、設定ファイル生成部２内の各記憶部内で指定されたコンフィグレーションを参照して、指定されたシステムＬＳＩの構成を網羅的に検証する検証ベクトルを生成する。ここで、システムＬＳＩの規模が大きい場合等には、限られた時間の中で検証を終えるために、基本となる構成から変更された部分を重点的に検証することが必要となる。そこで、指定されたオプション命令や指定されたサイズのキャッシュ等を重点的に検証するように、検証ベクトルは、命令セット記述に基づいて、各命令に対応する検証ベクトル郡を生成することが望ましい（図１１（ｂ）は検証ベクトル生成用のユーザ定義命令ファイルの一例を示す）。

【００５４】『性能評価部の構成』性能評価部５は、アプリケーションのコードサイズ、実行命令数、実行サイクル数、チップのゲートサイズおよび消費電力を見積る。以下では、この性能評価部による性能評価処理について詳しく説明する。

【００５５】・アプリケーションのコードサイズ評価
性能評価部５は、アプリケーションのコードサイズを評価することができる。

【００５６】今、アプリケーションとして図１２（ａ）に示すようなＣ言語のプログラムがあるとする。図１２（ａ）に示すアプリケーションは、システムＬＳＩ開発環境生成部４が生成するコンパイラを利用して、以下のようにコンパイルすることができる。

【００５７】＞ＳＬＩＤＥ＿ｃｃ　ｔｅｓｔ．ｃ
ここで、ＳＬＩＤＥ＿ｃｃ、ｔｅｓｔ．ｃはそれぞれ、コンパイラの起動コマンドおよびユーザが作成したアプリケーションのファイル名を示す。

【００５８】そして、コンパイルの結果得られるオブジェクトファイルからアプリケーションのコードサイズに関する情報を得ることができる。図１２（ａ）に示すＣプログラムの場合、そのコードサイズは６２となる。

【００５９】なお、ここではアプリケーションはＣ言語で記述されたものを例としてあげたが、システムＬＳＩ開発環境生成部４はアセンブラも生成することができるので、アプリケーションはアセンブリ言語で記述されたもの（あるいは混合されたもの）でも構わない。アプリケーションがアセンブリ言語で記述された場合は、アセンブルの結果としてオブジェクトファイルが得られる。また、アプリケーションプログラムはＲＯＭ上に配置されるため、ＲＯＭのサイズはアプリケーションプログラムのコードサイズおよび初期値付き変数のデータサイズに依存することになる。ＲＯＭのサイズは、通常、１２８[ＫＢ]や２５６[ＫＢ]のように２ｎ[ＫＢ]（ｎは自然数を示す）となるため、コードサイズがこの境界にある場合には、小さいサイズのＲＯＭに収まるようにアプリケーションを変更すれば、ＲＯＭサイズが小さくなり、コストの削減につながる。例えばコードサイズが１３０[ＫＢ]の場合、ＲＯＭサイズは２５６[ＫＢ]となるが、アプリケーションを改良してコードサイズが１２８[ＫＢ]以下になるようにすれば、ＲＯＭサイズは１２８[ＫＢ]になり、コストの削減につながる。

【００６０】また、システムＬＳＩ開発環境生成部４は、シミュレータを生成することができるので、生成されたシミュレータを利用して、実行ファイルをシミュレートすることもできる。シミュレータは、シミュレーション結果を表示するだけでなく、シミュレート時に実行される命令を１つずつカウントすることで、アプリケーション全体の実行命令数を計測することができる。例えば図１２（ａ）に示すプログラムのコンパイルの結果得られる実行ファイルの名前をｔｅｓｔとし、この実行ファイルｔｅｓｔをシミュレータ（起動コマンドＳＬＩＤＥ＿ｓｉｍ）にかけることで、

＞ＳＬＩＤＥ＿ｓｉｍ　ｔｅｓｔ
ｒｅｓｕｌｔ＝４５０
プログラム停止。実行命令数：７０４

のようにプログラムの実行結果（ｒｅｓｕｌｔ＝４５０）と実行命令数（７０４）を得ることができる。

【００６１】実行命令数は単に実行される命令を種類に関係なくカウントするだけで得られるので、短時間に測定することができる。これによりユーザは大まかなアプリケーションの性能に短時間に知ることができ、ＬＳＩ設計サイクルの短期化につながる。

【００６２】・アプリケーション実行時の開始から終了までに実行される命令数の総和（実行命令数）の評価
性能評価部５は、アプリケーション実行時のキャッシュミスの測定、実行される各命令のサイクル数の計測を行って、アプリケーション全体の正確なサイクル数を測定することができる。例えば、図１２（ａ）のプログラムのコンパイルの結果得られる実行ファイルの名前をｔｅｓｔとし、この実行ファイルｔｅｓｔをシミュレータ（起動コマンドＳＬＩＤＥ＿ｓｉｍ）にかけることで、

＞ＳＬＩＤＥ＿ｓｉｍ　ｔｅｓｔ
ｒｅｓｕｌｔ＝４５０
プログラム停止。実行サイクル数：１１９０

のようにプログラムの実行結果（ｒｅｓｕｌｔ＝４５０）と実行サイクル数（１１９０）を得ることができる。この性能見積もりにより、ユーザはＬＳＩの性能が余っているのか不足しているのかを知ることができ、コンフィグレーションの変更が容易となり、ＬＳＩ設計サイクルの短期化につながる。

【００６３】また、性能評価部５は、アプリケーションプログラム内に実行命令数計測の開始点と終了点を意味する命令を記述することにより、アプリケーションプロ

グラムの指定された２点間の実行命令数を測定する。

今、図１２（ａ）のプログラムの内側のｆｏｒループの実行命令数を計測しようとする場合、プログラムに図１２（ｂ）のように命令を書き加える。そして、コンパイラが図１３のプログラムをコンパイルすると、＿ＳＴＡＲＴと＿ＥＮＤが記述されたアドレスが保存され、シミュレート時に保存されたアドレス間の実行命令数を測定するようにしている。＿ＳＴＡＲＴ、＿ＥＮＤに続く括弧内の数字が＿ＳＴＡＲＴと＿ＥＮＤの対応関係を示している。図１２（ｂ）のアプリケーションのシミュレート結果は、図１３（ａ）に示すように、内側のループ（番号１の区間）の実行命令数が３７０であることが得られる。

【００６４】このように、測定区間を限定することにより、実行命令数測定時間を短くすることができる。特に、ループ変数の値によらず処理内容が変わらないループの１回の実行命令数が知りたい場合は、そのループを測定区間として指定し、ループが１回終わったところでシミュレートを終了させれば、測定時間の短縮につながる。これにより、アプリケーションの特定の場所だけの実行命令数を知ることができ、測定時間の短縮によるＬＳＩ設計サイクルの短縮化だけでなく、アプリケーションの改良箇所の探索にも有効な手段となる。

【００６５】・アプリケーション実行時の開始から終了までに実行されるサイクル数の総和（実行サイクル数）の評価

性能評価部５は、実行サイクル数の測定を実行することもできる。具体的には、区間指定の方法は上記の実行命令数の場合と同様であり、たとえば図１２（ａ）のプログラムの内側のｆｏｒループの実行サイクル数を測定しようとする場合、図１２（ｂ）に示すように、内側のｆｏｒループの前後に＿ＳＴＡＲＴと＿ＥＮＤを書き加える。この結果、シミュレート結果として図１３（ｂ）に示す結果が得られる。この処理により、実行サイクル数の測定時間の短縮につながるので、ＬＳＩ設計サイクルの短期化だけでなく、アプリケーションの改良個所の探索、さらにキャッシュの設定にも有効な手段となる。

【００６６】なお、ユーザが指定した区間だけでなく、指定した点を通過したときの実行命令数と実行サイクル数を出力するようにしても良い。これにより、ユーザはアプリケーションの実行開始時から任意の点までの実行命令数と実行サイクル数を知ることができる。例えば、ｍａｉｎ関数の中で呼び出される、ある関数内のループが終わった時点での実行命令数を知りたい場合を考える。この時、実施例４，５のような区間指定の実行命令数では、関数をまたがって区間を指定することができないため、アプリケーションの最初からの実行命令数を知ることができない。もし区間指定ではなく、プログラム上の任意の場所という指定ができれば、この問題は解消する。シミュレータは指定された点のアドレスを保持しておき、アプリケーション開始時からそのアドレス通過時までの実行命令数を出力するようにすればよい。

【００６７】まず指定されたアドレスが条件付分岐命令以外の命令、例えば加算を行うａｄｄ命令の場合を考える。シミュレータはこのａｄｄ命令を実行する前と後での、アプリケーション開始時からの実行命令数を出力できる。ａｄｄ命令の実行の前、後の選択はユーザが行うことができる。

【００６８】次に指定されたアドレスが条件付分岐命令、例えばｂｅｑ命令の場合を考える。ｂｅｑ　Ｒ１，Ｒ２，ＬＡＢＥＬ１という命令は、レジスタＲ１とＲ２の値が等しいときにＬＡＢＥＬ１に記述されたアドレスに分岐し、等しくないときは分岐命令の次の命令を実行する（ここでは分岐命令の遅延は無いものと考えている）。シミュレータはこのｂｅｑ命令の条件が成立したときと成立しないときの両方の、アプリケーション開始時からの実行命令数が出力できる。条件成立／不成立の選択はユーザが行うことができる。

【００６９】・消費電力、ゲートサイズの評価

性能評価部５は、システムＬＳＩ開発環境生成部３から出力されるＲＴＬ記述を参照して、ワットウォッチャ（WattWatcher）等の市販ツールによって消費電力あるいは消費電力に換算可能な数値を抽出する。また、ＲＴＬ記述を利用して、ゲートサイズあるいはゲートサイズに換算可能な数値を抽出する。ゲートサイズに関する情報を抽出することにより、最適なチップ面積を決定でき、ＬＳＩ製造時のコストを抑えることにつながる。

【００７０】なお、性能評価部５は、システムＬＳＩ開発環境生成部３が生成したＲＴＬ記述、コンパイラ、シミュレータ、検証環境を利用して、キャッシュのミス率（あるいはヒット率）を抽出することもでき、例えば、目的アプリケーションに対するキャッシュ性能の見積り結果として図１３（ｃ）に示すような値を得ることができる。この結果、キャッシュサイズに制約がなく、目的アプリケーションの実行時間を短くしたい場合はキャッシュミス率が一番低い１６Ｋｂｙｔｅｓをキャッシュサイズとして設定することになる。また、キャッシュサイズに制約がある場合は、キャッシュミス率の変更に伴う目的アプリケーションの実行時間の変更とのトレードオフを考えることになる。

【００７１】また、性能評価部５は、生成したＲＴＬ、コンパイラ、シミュレータ、検証環境に基づいて得たＲＴＬのゲートサイズの消費電力などの見積もり情報や、コンパイル、シミュレートの結果わかるアプリケーションの実行命令数や実行サイクル数といった見積もり情報を内部に保持していることが望ましい。そして、これらの情報をユーザが閲覧しようとする場合、すべての情報を閲覧することも可能であるが、本ツールではユーザが見たいと思う項目（最優先項目）を指定することで自動的にその項目だけを閲覧することができる。たとえば、

ユーザがチップサイズを見積もり情報として知ろうとした場合、最優先項目としてチップサイズを指定すれば、本ツールは生成したＲＴＬからチップサイズを計算してユーザに提出する。アプリケーションの実行サイクル数を知ろうとした場合、シミュレート結果から判明した実行サイクル数をユーザに提出する。

【００７２】このような構成によれば、ユーザは知りたい見積もり情報を条件として指定することで、自動的にその見積もり情報が得られることになり、見積もりの判断が迅速に行えるようになる。

【００７３】なお、性能評価部５は，ＲＴＬシミュレーションを行った結果のサイクル情報をシミュレータの入力として、変換できる。例えば、メモリアクセスといっても、メモリには、いくつか種類がある。ＲＯＭやＲＡＭなどではアクセスサイクル数が異なってくる。このような時、シミュレータでは、シミュレータを生成する際に大まかなサイクル数を設定することはできるが、それは実際の値に沿ったものではない。その値の修正を行うためにＲＴＬでシミュレーションを実行した結果のサイクル情報を利用する。シミュレータは、ＲＴＬシミュレーションのサイクル情報または、サイクル情報をシミュレータの入力フォーマットに変換した情報を読み取り、シミュレータ内部の値の変更が可能になる。これにより、より実際に近い値の見積もりを行うことができる。

【００７４】『終了判定部の構成』終了判定部６は、性能評価部５による性能評価結果に基づいて、設計されたシステムＬＳＩがユーザが予め設定した目標性能を満たしているか否かを判別する。

【００７５】そして、目的性能を満たしていない場合に

（テンプレート）

サイクル数
が多い　→メモリアクセスの
サイクル数が多い　→キャッシュの
ミス率が高い　→ＡＮＳ：キャッシュ
サイズを増やす

→その他の理由（省略）

このように、メモリ領域をアクセスサイクルで分類し、領域ごとに統計情報を持つことにより、より正確な見積もりが得ることができ、また、目的アプリケーションの実行時の統計情報を元にした評価を行うことにより、より適正な可変値の設定値を導出できる。また、自動的に可変値の設定値を導出できる手段を持つことにより、開発時間を短縮することができる。

【００８１】また、変更項目設定部７は、性能評価のための統計情報として、目的アプリケーションを実行した結果のオプション命令の統計情報を有することが望ましい。そして、この統計情報を利用して、使われていない命令をはずしたオプションの構成に再編成をすることができる。例として、３２ビットの積和演算のオプション命令をつけてＬＳＩを構成した場合を考える。この構成

は、変更項目設定部５が、性能評価部５による評価結果に基づいて、次回以降の設定値を導出する。

【００７６】『変更項目設定部の構成』変更項目設定部７は、性能評価部５による評価結果に基づいて、次回以降のコンフィグレーションの設定値を導出する。性能評価のための統計情報として、メモリ領域に対するアクセス統計、アクセスサイクル数を有している。また、キャッシュがついている場合は、キャッシュのヒット率も統計情報として提供される。

【００７７】目的アプリケーションを実行した際の統計情報を性能評価のための情報として利用することにより、目的アプリケーションの実行の特徴に沿った可変値の設定値を導出することができる。

【００７８】メモリ領域（ＲＯＭ、ＲＡＭ、キャッシュなど）は、アクセスサイクル数に基づき分類され、メモリ領域のアクセス統計とサイクル数計算部分とは分離されている。

【００７９】アクセス統計とサイクル数計算を切り離しているのは、サイクル数補正の演算は必要なときのみに行われるようにしたためである。ここで、キャッシュ付システムにおいて１回では目的性能が得られなかった場合を考える。性能評価の手段として、目的アプリケーションのサイクル数が多い場合に、その部分を減らしたい場合に、アクセス統計の情報を利用し、統合環境が持っている、サイクル数を減らすためのテンプレートに従い、キャッシュのサイズを大きくするなどの可変値の設定を行う。

【００８０】

で目的アプリケーションの実行させ、実行時の統計情報をとる。その結果、３２ビットの積和演算のオプション命令は使用しないことがわかり、そのオプション命令をＯＦＦにした構成を性能評価からのフィードバックとして、設定を変更し、再構築を行う。このように使用していない命令を除くための設定や再構築が自動的に行えることにより、開発時間を短縮し、かつコストの削減が可能になる。

【００８２】さらに、変更項目設定部７は、評価結果により次回以降の可変値の設定値を導出する手段をもち、また、性能評価のための統計情報として、目的アプリケーションを実行した結果のオプション命令の統計情報を有することが望ましい。これにより、この統計情報を利用して、あらかじめ定められた出現回数または出現率以

下の命令をはずしたオプションの構成に再編成することができる。例として、３２ビットの積和演算のオプション命令をつけてＬＳＩを構成して場合を考える。この構成で目的アプリケーションの実行させ、実行時の統計情報をとる。その結果、３２ビット積和演算のオプション命令の使用頻度があらかじめ定められた出現回数または出現率以下であることがわかったので、そのオプション命令をＯＦＦにした構成を性能法科からのフィードバックとして、再構築を行う。コンパイラは、そのオプション命令を使用せずに、コンパイルを行う。このように命令の再構築、コンパイラの再構築、また、自動的な設定の変更が行えることにより、開発時間を短縮し、かつコストの削減が可能になる。

【００８３】さらに又、変更項目設定部７は、　性能評価により自動的に可変項目の設定変更を行った後で、再度目的アプリケーションを実行して、設定変更の正当性を判断する手段を有することが望ましい。また、性能評価により自動的に可変項目の設定変更を行った後で再度目的アプリケーションを実行して、設定変更が有効でないと判断した場合には、別の設定に変更して、これを、有限実行回数の施行を繰り返し、得られた結果の中で適切な設定を自動的に選択すると良い。

【００８４】『入出力インタフェイス部の構成』入出力インタフェイス部８は、システムＬＳＩ開発装置１と入力部１０、出力部１１との間の情報の入出力処理を支援する機能を有し、例えば、グラフィカルユーザインタフェイス（Graphical User Interface GUI）の利用が考えられる。この入出力インタフェイス部８によれば、コンピュータに不慣れなユーザであっても、入出力インタフェイス部８からの指示に従って、容易にシステムＬＳＩの開発を行うことができる。

【００８５】『制御部の構成』制御部９は、入出力インタフェイス部８を介したユーザからの指示に従って、システムＬＳＩ開発装置１内の構成要素の制御を行う。

【００８６】《システムＬＳＩ開発装置の動作》次に、上記のシステムＬＳＩ開発装置を用いたシステムＬＳＩ開発処理について説明する。

【００８７】上記のシステムＬＳＩ開発装置を用いてシステムＬＳＩ開発を行う際には、ユーザは、まず始めに、キャッシュサイズ等、性能に関して最大のコンフィグレーションをシステムＬＳＩ開発装置１に対して入力する（Ｓ１０１）。コンフィグレーションが入力されると、装置１内のシステムＬＳＩ開発環境生成部４が、コンフィグレーションの可能な全ての組み合わせに対応したツールを生成し（Ｓ１０２）、ツール生成完了に応じてユーザはアプリケーションを実行させる（Ｓ１０３）。そして、アプリケーション実行後、性能評価部５が、アプリケーションの実行結果を参照して、アプリケーションの性能を評価する（Ｓ１０４）。

【００８８】性能評価部５による性能評価が完了すると、次に、終了判定部６が、最初のコンフィグレーションによる性能が所望の基準に達しているか否かを判別し（Ｓ１０５）、達している場合には、性能を満足する最小のシステム構成を抽出し（Ｓ１０８）、検証環境およびドキュメントをユーザに対して出力する（Ｓ１０９）。ここで、ドキュメントには、指定されたコンフィグレーションを確認するための項目についての記述が含まれているものとする。

【００８９】一方、性能が足りていない場合には、ユーザは、ユーザ定義のモジュールを設計し、開発ツール内に組み込み（Ｓ１０６）、ユーザ定義モジュールを利用するようにアプリケーションを書き換える（Ｓ１０７）。そして、再びアプリケーションを実行し、書き換えたアプリケーションが所望の性能を満たしているか否かを判別する。

【００９０】（ユーザ定義モジュールの追加する処理の例）ここで、ユーザ定義モジュールとしてＤＳＰを追加する処理を例として挙げ、上記のユーザ定義モジュールを追加する際の処理について説明する。なお、この例においては、アプリケーションとして、メモリよりデータを読み出し、計算を行い、結果を書き戻すという処理を繰り返し実行するものを考える。

【００９１】この例においては、プロセッサのみを用いてアプリケーションの処理を実行すると、アプリケーションでは、プロセッサの命令を使用して、図１５(a)に示すようなプログラムを実行する。しかし、このままでは目標性能を達成することはできないので、ユーザ定義モジュールとして、計算を行うＤＳＰを追加する。ここで、ＤＳＰは制御バスによって制御され、コアのプロセッサは制御バスにＤＳＰの制御のための数値を書き込めば良い。なお、この処理はＤＳＰ命令を定義して使用することも可能であるが、ここでは、制御バスでＤＳＰの実行を制御することとする。なお、図１５（ａ）中の_culcは演算処理を行うサブルーチンを示す。

【００９２】また、ユーザ定義モジュールとして追加されたＤＳＰは、図１５（ｂ）に示すプログラムに従って動作する。すなわち、制御バスのcntlbus_addr1のアドレスに、計算用のデータの開始アドレスをおき、cntlbus_addr2に計算用のデータの終了アドレスを置く。また、cntlbus_addr3は、計算結果を収めるメモリの位置の値を置く。cntlbus_addr4では、このアドレスに書き込むことにより、ＤＳＰの動作の制御が行われる。具体的には、このアドレスのあるビットに１を書き込むことにより、ＤＳＰの処理を開始したり、停止したりする。処理が開始されると、ＤＳＰは、開始アドレスや終了アドレス、書き込み先のアドレスを決められた場所（cntlbus_addr1, cntlbus_addr2, cntlbus_addr3）に読みにいき、処理を開始する。

【００９３】ＤＳＰのようなユーザ定義モジュールを追加する方法としては、以下の２つの方法が考えれる。す

なわち、（１）ユーザ定義モジュールの仕様として、動作記述とＲＴＬ記述の双方を作成し、ユーザ定義モジュール・ユーザ定義命令記憶部３内に格納する、又は、（２）動作記述のみを作成し、ＲＴＬ生成部４１内の高位合成処理部４１ｅにおいて、動作記述からＲＴＬ記述を生成する。

【００９４】ここで、図１５（ｃ），１６を参照して、上記の動作記述について説明する。図１５（ｃ）内のDSP_HWEngineは、HWEngineの性質とcontrolbusの性質を有し、クラスの定義をすることにより、動作モデルは、DSPをHWEngineと認識する。また、コントロールバスにＤＳＰ開始等の書き込みをした場合には、図１６に示すプログラムに従って、DSP_HWEngineのwrite_controlbusが呼び出され、ＤＳＰの処理を開始する。また、ユーザがＤＳＰに対して命令を記述したい時には、ＭＰＩのコンフィグレーション時にユーザ定義の命令を記述市、コンフィグレーション情報を受け取ったシステムＬＳＩ開発装置が命令定義ヘッダファイルを作成し、コンパイラに与える。コンパイラは、コンパイル処理を実行し、ユーザ定義命令の入った実行オブジェクトを出力する。なお、シミュレータやＲＴＬにおいてその動作を実行するためには、その動作記述を与える必要性がある。また、図１６に示す動作記述では、DSP_HWEngineクラスの中にdo_command関数に命令毎に記述を行う。

【００９５】上記のようにして追加されたユーザ定義モジュールを用いて性能評価を行う。そして、追加したモジュールが性能を満たしていれば終了。満たしていなければ、さらにユーザ定義モジュールを作成し、追加する。ソフトウェアの性能評価は、ユーザアプリケーションを実行し、目標性能を満たしているか否かを調べる。ハードウェアの性能評価は、コアプロセッサとユーザ定義モジュールとを接続したＲＴＬ記述を論理合成し、ゲート数計算、タイミング解析、消費電力解析等を実行する。ハードウェア検証は、ユーザが手書によって与えたベクトル、又は、検証用のベクトルを生成するツールを使用して作成したベクトル等を用いて、コアとユーザモジュールから成る動作モデル（シミュレータ）のベクトル実行結果とＲＴＬシミュレーションの結果が一致しているかどうかで行う。

【００９６】（マルチプロセッサ構成の処理例）以上の説明においては、システム内にプロセッサを１つ設ける例であったが、当然のことながら、システム内にプロセッサを複数設け、マルチプロセッサ構成を構築することが可能である。このマルチプロセッサ構成においては、オプション命令やキャッシュサイズ等の可変設定項目は各プロセッサ毎に独立して設定することが可能である。以下では、図１７，１８を参照して、マルチプロセッサ構成の処理例について説明する。

【００９７】圧縮されたデータの復元処理やデータのディスプレイ表示処理等、近年の傾向としては、ハードウェアで行っていた処理もコスト面や速度面からソフトウェアで代用できるようになり、ソフトウェアの処理の割合が増し、複雑化してきている。このような、複雑なアプリケーションがターゲットである場合、１つのプロセッサにユーザ定義モジュールを定義したり、コプロセッサをつけたりしても、目標性能を満たしそうにない時は、マルチプロセッサ構成を採用する。また、速度向上のためにコプロセッサをつけた場合でも、コプロプロセッサへのロード等、コアプロセッサが行う処理は多い。また、明らかに１プロセッサでは処理不可能な場合もある。そのような場合は、複数のプロセッサでアプリケーションを実行することを考える。

【００９８】ここで、圧縮されたデータストリームを（１）受信、（２）復元、（３）ディスプレイ表示用に加工、（４）データ出力するというアプリケーションを考える。ストリームの処理は、処理時間が決まっているので、まず、処理速度が大切である。これらを１つのプロセッサで行った場合、複数の処理＋コプロセッサやユーザ定義モジュールへの制御も行わなければならず、どうやっても目標性能を得ることができない。このような場合、処理毎にプロセッサを割り当て、処理を分散させると良い。例として、プロセッサ１で（１）と（４）の処理、プロセッサ２で（２）の処理、プロセッサ３で（３）というように処理を分散させることができる。また、オプション命令やキャッシュ等も、例えば、プロセッサ１は、命令キャッシュ無し、データキャッシュ無し、オプション命令無し、プロセッサ２は、命令キャッシュ２ＫＢ、データキャッシュ１ＫＢ、ユーザ定義モジュールとしてのＤＳＰ付き、コプロセッサ無し、プロセッサ３は、命令キャッシュ２ＫＢ、データキャッシュ無し、コプロセッサ有りというように、プロセッサ毎に設定することができる。すなわち、今まで述べてきた、１つのプロセッサへの可変項目設定を複数のプロセッサについても行うことができる。そして、３つのプロセッサを使ってそれぞれの処理を行うアプリケーションで、それぞれのプロセッサに対する、最小構成を求めることにより、最適なＬＳＩを構築することができる。例えば、最初、全てのプロセッサの命令キャッシュ、データキャッシュを４ＫＢとしていたが、性能評価の結果、上記のような構成で目標性能が得られということがわかる等、それぞれの可変項目について、性能評価からのフィードバックを得ることができる。

【００９９】マルチプロセッサ構成のＬＳＩは、シングルプロセッサ構成の時と同様、コンフィグレーションを指定することにより、システムＬＳＩ開発環境を生成することができる（ただし、ユーザ定義モジュールがある場合には動作記述を作成する必要がある）。コンパイラはマルチプロセッサに対しての変更はないが、個々のプロセッサに対して１対１にコンパイラが生成される。シミュレータ、デバッガ等は、マルチプロセッサに対応し

ているものが生成される。システムＬＳＩ開発環境がどのようにしてマルチプロセッサのシミュレータを生成できるかであるが、これも、シングルプロセッサの時と同様、プロセッサの設定や生成をコア毎にテンプレートを使用して設定用の関数を生成する（図１７，１８に示すプログラムコードがシミュレータ生成用のテンプレートを使用して生成されたものである。なお、図１７(a)に示すコードはシミュレータのコア用のクラスの例である）。コア毎に生成された設定関数（図１７（ｂ））をモジュール組み立ての関数（図１８(a)）がコールする。ここもコンフィグレーションによって呼ばれる関数の数が異なってくるので、テンプレートを使用して生成される。また、シミュレータを実行する部分は、図１８（ｂ）に示すように、１クロック又は１ステップ（１命令）実行のためコールされる関数中に、それぞれのプロセッサの１クロック又は１ステップ実行部分の関数がコールされるようにされている。

【０１００】また、アプリケーションでは、プロセッサ間の通信処理等も必要になるが、共有メモリやプロセッサ間ハードウェア割り込みや、ハードウェアセマフォ等がある。上記のアプリケーションでは、計算が終了したデータをメモリに書き込み、他のコアとやり取りしたり、処理が終了したことをあるコアに知らせるために、そのコアに対してハードウェア割り込みを起こしたり、資源の競合を防ぐために、セマフォを利用したりする。ＲＴＬもシミュレータと同様にテンプレートがあり、コアの個数の応じて、ＲＴＬ記述が生成される。

【０１０１】このように、この実施の形態のシステムＬＳＩ開発装置においては、システムＬＳＩの設計上のコンフィグレーションを１つのデータベース内に保持し、このデータベース内の情報を参照して、ＲＴＬ記述、開発ツールおよび検証環境の全てを一貫して生成するので、コンフィグレーションの変更に伴う再設計に要する労力および時間を大幅に削減することができる。また、このような構成によれば、設計→評価のプロセスを比較的短時間に繰り返し、ユーザの目的に適した構成のシステムを容易に得ることが可能となる。さらには、最初に最大構成の設定から評価を行うことにより、ユーザ定義モジュールの必要性を判断し、システムＬＳＩの設計に早期に着手することが可能となる。

【０１０２】《その他の実施の形態》本発明の実施の形態のシステムＬＳＩ開発装置には、いわゆる汎用機、ワークステーション、ＰＣ（Personal Computer）、ＮＣ（Network Computer）等が含まれ、例えば、図１８に示す構成のような概観を有し、フロッピー（登録商標）ディスクドライブ５２および光ディスクドライブ５４を備えている。そして、フロッピーディスクドライブ５２に対してはフロッピーディスク５３、光ディスクドライブ５４に対しては光ディスク５５を挿入し、所定の読み出し操作を行うことにより、これらの記録媒体に格納されたプログラムをコンピュータシステム内にインストールすることができる。また、所定のドライブ装置５７を接続することにより、例えば、メモリ装置の役割を担うＲＯＭ５８や、磁気テープ装置の役割を担うカートリッジ５９を用いて、インストールやデータの読み書きを実行することもできる。

【０１０３】また、本発明の実施の形態のシステムＬＳＩ開発装置は、プログラム化しコンピュータ読取り可能な記録媒体に保存しても良い。そして、システムＬＳＩ開発処理を行う際は、この記録媒体をコンピュータシステムに読み込ませ、コンピュータシステム内のメモリ等の記憶部にプログラムを格納し、システムＬＳＩ開発プログラムを演算装置で実行することにより、本発明のシステムＬＳＩ開発装置およびその方法を実現することができる。なおここで言う記録媒体とは、例えば、半導体メモリ、磁気ディスク、光ディスク、光磁気ディスク、磁気テープ、伝送媒体等のプログラムを記録することができるようなコンピュータ読取り可能な記録媒体等が含まれる。

【０１０４】《実施の形態の効果》以上述べてきたように、この実施の形態のシステムＬＳＩ開発装置によれば、オプション命令、ユーザ定義モジュールおよびマルチプロセッサ構成の少なくとも１つを含む可変項目定義情報の可能な全ての組み合わせについて、オプション命令が定義されたプロセッサを含むシステムＬＳＩのハードウェア、検証環境、開発ツールを一貫して網羅的／複数並列に作成するので、アプリケーションに最適なシステムＬＳＩを短期間に開発することが可能となる。

【０１０５】また、１箇所で指定されたコンフィグレーションを各ツールで共有し、コンフィグレーションの可能な組み合わせについて、システムＬＳＩのハードウェア、検証環境、開発ツールを一貫して網羅的／複数並列に作成するので、ツール間の連携をとることが可能となり、システムＬＳＩの開発期間を短縮することができる。また、マルチプロセッサシステムを構築する場合には、全てのハードウェアのローカルメモリ情報を一括して保持し、各プロセッサのローカルメモリマップを他のプロセッサのローカルメモリへのアクセス領域を含めて生成するので、システムＬＳＩ中に複数のプロセッサを含めることが可能となる。

【０１０６】また、コンフィグレーションの各組み合わせについて、並列に性能評価を実行するので、人手による各ツール出力の評価集約を行う必要がなくなり、システムＬＳＩの開発期間を短縮することが可能となる。また、アプリケーションのコードサイズ、実行命令数、実行サイクル数、チップのゲートサイズ、消費電力を見積もる機能を有し、見積もった性能と目標性能とを比較して評価するので、性能評価に要する労力を軽減し、システムＬＳＩ開発期間の短縮を実現することが可能となる。

【０１０７】また、コンフィグレーション項目の設定を対話的に行うことができるので、ユーザは設定ファイルの文法を意識せずにコンフィグレーションを設定することが可能となり、直接設定ファイルを記述する場合と比べて短時間に設定を行うことができる。また、全てのコンフィグレーション項目の設定を対話的に行うことができるので、コンフィグレーション項目の設定のし忘れの防止にもなる。

【０１０８】また、コンパイラカスタマイズ部４３がコンパイラを生成するので、目的アプリケーションのコードサイズを知ることができ、システムＬＳＩの開発期間を短縮することが可能となる。

【０１０９】また、シミュレータカスタマイズ部４２がシミュレータを生成するので、目的アプリケーションの性能を短時間で知ることができ、システムＬＳＩの開発期間を短縮することができる。

【０１１０】また、性能評価部５がアプリケーションの実行サイクル数を抽出するので、ユーザは、ＬＳＩの性能が足りているのか、不足しているのかを容易に知ることができ、コンフィグレーション項目の速やかな変更につながり、システムＬＳＩの開発期間を短縮することができる。

【０１１１】また、性能評価部５がゲートサイズを抽出するので、最適なチップ面積を決定し、ＬＳＩ製造時のコストを抑えることが可能となる。

【０１１２】さらに、ユーザが指定した可変値に基づいてコンフィグレーションの設定およびその性能評価を行うことができるので、複数の設定値に対応した結果を同時に得ることが可能となり、ＬＳＩ開発サイクルの短縮を実現することができる。

【０１１３】さらにまた、システムＬＳＩ開発上の最優先項目を指定することができるので、構成を考えることなく、優先項目に適した構成の見積を得ることができる。

【０１１４】また、メモリ領域をアクセスサイクルで複数の領域に分類し、各領域に対して統計情報を持たせるので、より正確な見積を得ることができる。

【０１１５】また、目標性能を設定することができるので、目標性能に沿った見積を速やかに導出することができる。また、アプリケーションが目標性能に到達しなかった場合には、アプリケーションの実行時に記録した情報に基づいてコンフィグレーション項目を変更（フィードバック）するので、目標性能にあった構成を抽出することが可能となる。さらに、自動変更されたコンフィグレーション項目に基づいて開発環境、検証環境を再度生成し、アプリケーションの実行性能を測定しし、目標性能と比較することで設定変更の妥当性を判断することができる。さらにまた、妥当性を判断した結果、設定変更が妥当でないと判断された場合には、再度、コンフィグレーション項目を自動的に変更、アプリケーションの実

行性能の再測定、コンフィグレーション項目の設定変更の妥当性を判断するので、変更を有限回数行うことができる。これにより、システムＬＳＩ開発の工期を短縮することができる。

【０１１６】

【発明の効果】本発明によれば、最適な性能のＬＳＩを開発設計することができる。

【図面の簡単な説明】

【図１】本発明の実施の形態のシステムＬＳＩ開発装置の構成を示すブロック図である。

【図２】本発明の実施の形態の設定ファイル生成部の構成を示すブロック図である。

【図３】本発明の実施の形態のコンフィグレーションファイルを示す図である。

【図４】本発明の実施の形態のグローバルメモリマップ、ローカルメモリマップおよびユーザ定義命令を示す図である。

【図５】本発明の実施の形態のシステムＬＳＩ開発環境生成部の構成を示すブロック図である。

【図６】本発明の実施の形態のＲＴＬ生成部の動作を説明するための図である。

【図７】本発明の実施の形態のＲＴＬ生成部の動作を説明するための図である。

【図８】本発明の実施の形態のシミュレータカスタマイズ部の動作を説明するための模式図である。

【図９】本発明の実施の形態のシミュレータ起動オプションファイル、デバッガ起動オプションファイルおよび機械命令関数宣言ヘッダファイルを示す図である。

【図１０】本発明の実施の形態のコンパイラカスタマイズ部の動作を説明するための模式図である。

【図１１】本発明の実施の形態のコンパイラ起動オプションファイルおよび検証ベクトル生成用ユーザ定義命令ファイルを示す図である。

【図１２】本発明の実施の形態のアプリケーションプログラムを示す図である。

【図１３】実行命令数、実行サイクル数およびキャッシュミス率を示す図である。

【図１４】本発明の実施の形態のシステムＬＳＩ開発装置を用いたシステムＬＳＩ開発処理を示すフローチャート図である。

【図１５】アプリケーションのプログラム例および動作記述例を示す図である。。

【図１６】動作記述の例を示す図である。

【図１７】プログラムコードの例を示す図である。

【図１８】プログラムコードの例を示す図である。

【図１９】本発明の実施の形態のシステムＬＳＩ開発装置の概観を示す模式図である。

【符号の説明】

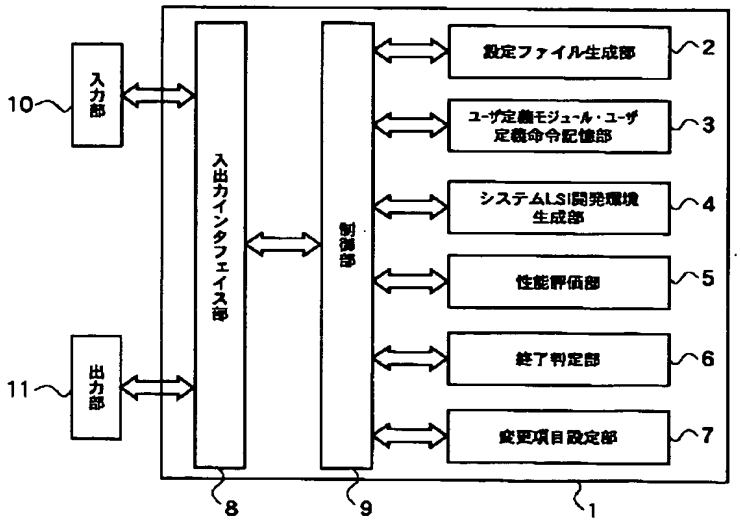
１　システムＬＳＩ開発装置

２　設定ファイル生成部

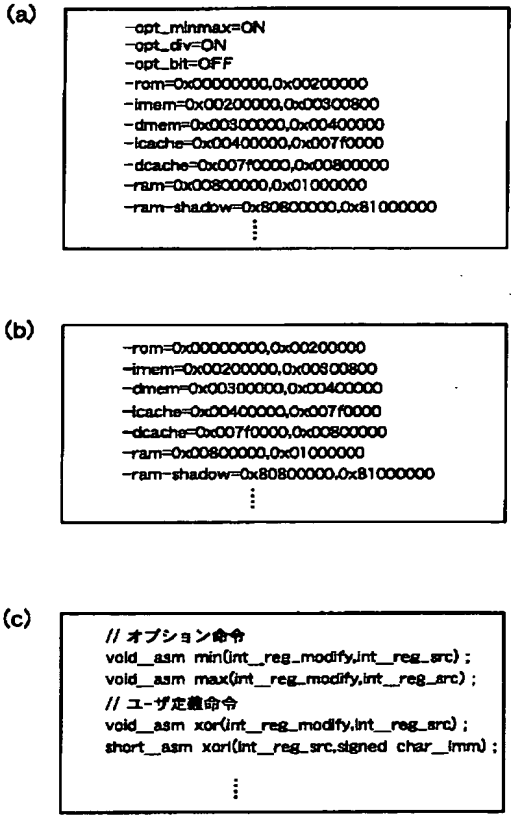
３　ユーザ定義モジュール・ユーザ定義命令記憶部３
４　システムＬＳＩ開発環境生成部
５　性能評価部
６　終了判定部
７　変更項目設定部
８　入出力インタフェイス部
９　制御部
１０　入力部
１１　出力部
２１　入力解析部
２２　ローカルメモリマップ生成部
２３　オプション情報記憶部
２４　デバイス構成記憶部
２５　キャッシュ構成記憶部
２６　ローカルメモリマップ記憶部
４１　ＲＴＬ生成部
４１ａ、４１ｂ　ＲＴＬテンプレート
４１ｃ　コアＲＴＬ記述
４１ｄ　ブロック接続部
４１ｅ　高位合成処理部
４２　シミュレータカスタマイズ部
４２ａ　オプション命令・Ｃモデルテンプレート

４２ｂ　シミュレータテンプレート
４２ｃ　再コンパイル部
４２ｄ　起動オプション情報抽出部
４２ｅ　組み合わせ処理部
４３　コンパイラカスタマイズ部
４３ａ　オプション命令情報抽出部
４３ｂ　オプション命令・機械命令関数宣言テンプレート
４３ｃ　機械命令関数宣言抽出部
４３ｄ　結合処理部
４４　アセンブラカスタマイズ部
４５　検証ベクトル生成部
５０　コンピュータシステム
５１　ディスプレイ
５２　フロッピーディスクドライブ
５３　フロッピーディスク
５４　光ディスクドライブ
５５　光ディスク
５６　キーボード
５７　ドライブ装置
５８　ＲＯＭ
５９　カートリッジ

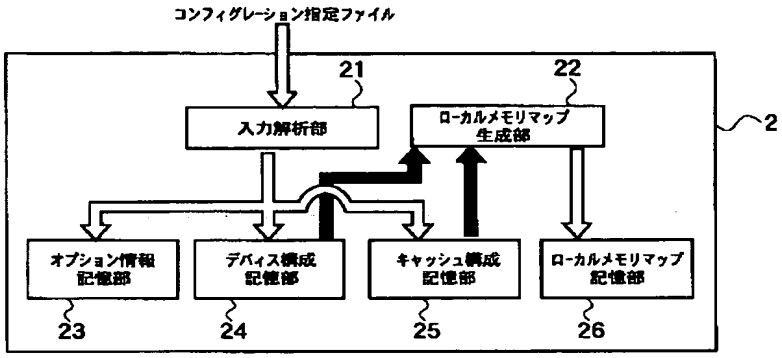
【図１】



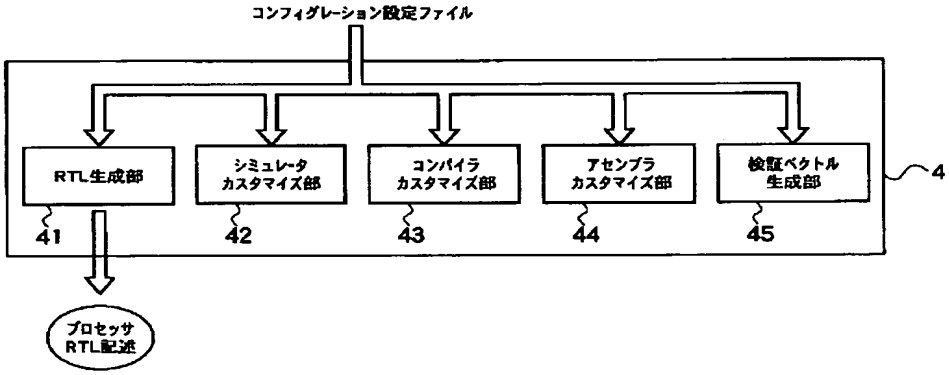
【図９】



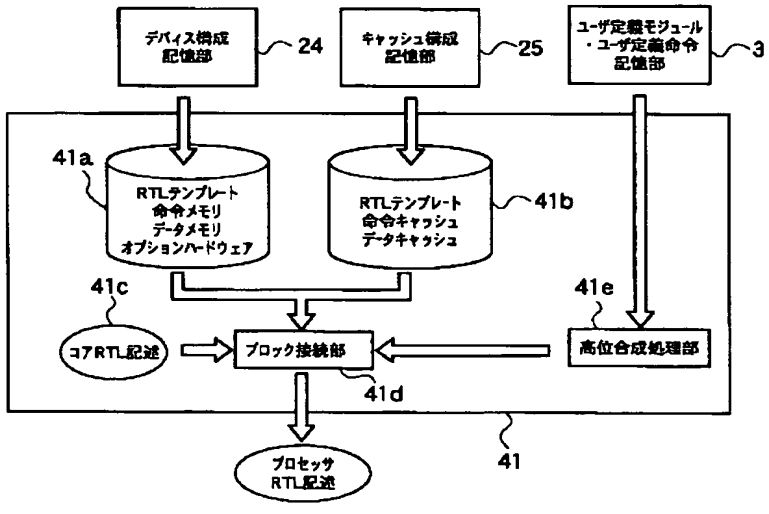
【図２】



【図５】



【図６】



41：RTL生成部

【図３】

```
//configuration data
CHIP_NAME="SimpleChip1" ; // チップ名
FREQUENCY=200 ;          // 周波数。単位 MHz。

//プロセッサ1
P_MODULE[Processor1]{
        P_ENGINE{
                // プロセッサコアの仕様
                CORE{
                        ID=0 ;  // コアID
                        // オプション命令
                        DIV=ON ;    //除算命令
                        MINMAX=ON ;   //最大/最小値命令
                        BIT=ON ;      //ビット操作命令
                };
                IMEM{
                        SIZE=4 ;        //KB
                };
                DMEM{
                        SIZE=2 ;        //KB
                BIU{  //バスインターフェイスユニット
                        BUS_SIZE=64 ;
                };
                INTC{   //割り込みコントローラ
                        CHANNEL_BITW=16 ; //未確定
                        LEVEL=15 ; //未確定
                };
                DSU{ //デバッグユニット
                        INST_ADDR_BREAK_CHANNEL=1 ;
                        DATA_ADDR_BREAK_CHANNEL=1 ;
                };
        };
};
//プロセッサ2
P_MODULE[Processor2]{
        P_ENGINE{
                CORE{
                        ID=1 ;  // コアID
                        // オプション命令
                        DIV=ON ;    除算命令
                        MINMAX=OFF ;   //最大/最小値命令
                        BIT=OFF ;      //ビット操作命令
                };
                IMEM{ //命令RAM
                        SIZE=8 ;        //KB
                };
                DMEM{ //データRAM
                        SIZE=20 ;       //KB
                };
                ICACHA{ //命令キャッシュ
                        SIZE=4 ;        //KB
                };
                DCACHE{  //データキャッシュ
                        SIZE=8 ;        //KB
                };
        };
        //コプロセッサ
        COPRO[Cop1]{
                IS_VLIW=YES ;  //VLIW型コプロ。
                VLIW_BTW=32 ;  //命令長32ビット
                ISA_DEFINE="cop1.isa" ;  //ISA定義、別ファイル
                RTL     ="cop1.v"        //RTL記述、別ファイル
        };
 //ユーザ定義モジュール
 UCI{
        ISA_DEFINE="uci1.uci" ; //ISA定義、別ファイル
        RTL="uci1.v" ; //RTL記述、別ファイル
        SIM="ucimode1.c" ;      //ハードウェアcモデル、別ファイル
 };
};
//グローバルマップ
GLOBAL_MAP="schip1.map" ;  //別ファイルを指定

//end of file
```

【図１２】

(a)

```
int  x=0 ;

main( )  {

      int  i , j ;
      for(i=0 ; i<10 ; i++)
          for(j=0 ; j < 5 ; j++)
              x+=i*j ;
      printf("result=%d ¥ n", x) ;
      }
```

(b)

```
int  x=0 ;
main( )   {
      int  i , j ;
      for(i=0 ; i<10 ; i++){
__START(1)
          for(j=0 ; j < 5 ; j++)
              x+=i*j ;
__END(1)
          }
      printf("result=%d ¥ n", x) ;
      }
```

【図４】

(a)

```
//start : size : name : cache(opt) : shadow_original_start(opt) : scope : type : read_write(opt)
0X0000_0000 : 0X0020_0000 : RAM1 :          : : global : memory : ro  ;
0X0080_0000 : 0X0080_0000 : RAM2 : Cache    : : global : memory : rw ;
0X8080_0000 : 0X0080_0000 : RAM3 :          : : global : memory : rw ;
```

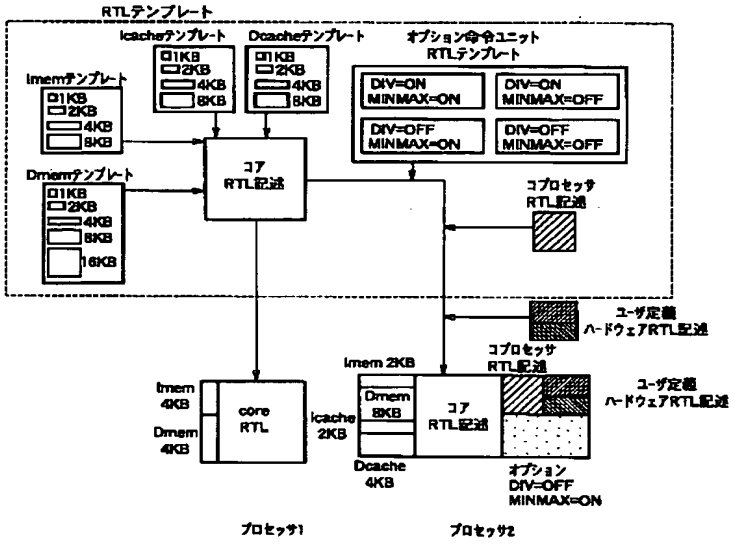
(b)

```
:RAM1     :0X00000000 : 0X00200000 :        :global : ro::      memory:
:imem0    :0X00200000 : 0X00002000 :        : local : rw::        imem:  in mm
:imem1    :0X00202000 : 0X00002000 :        : local : rw::        imem:  in mm
:dmem0    :0X00208000 : 0X00004000 :        : local : rw::        dmem:  in mm
:dmem1    :0X0020c000 : 0X00004000 :        : local : rw::        dmem:  in mm
:icahe_dat :0X00300000 : 0X00004000 :       : local : rw:: icache_data:  in mm
:icahe_tag :0X00310000 : 0X00004000 :       : local : rw::  icache_tag:  in mm
:RAM2     :0X00800000 : 0X00800000 : cache :global : rw::      memory:
:RAM3     :0X00808000 : 0X00800000 :        :global : rw::      memory:
```

(c)

```
#命令のニーモニック、オペコードと引数を宣言
void xor(int__reg_modify,int__reg_src) ;
{
        code16='0000_0010_0000_0000' ;
}
short xori(int__reg_src,signed char__imm) ;
{
        code16='0000_0011_iiii_iiii' ;
}
```

【図７】



【図１８】

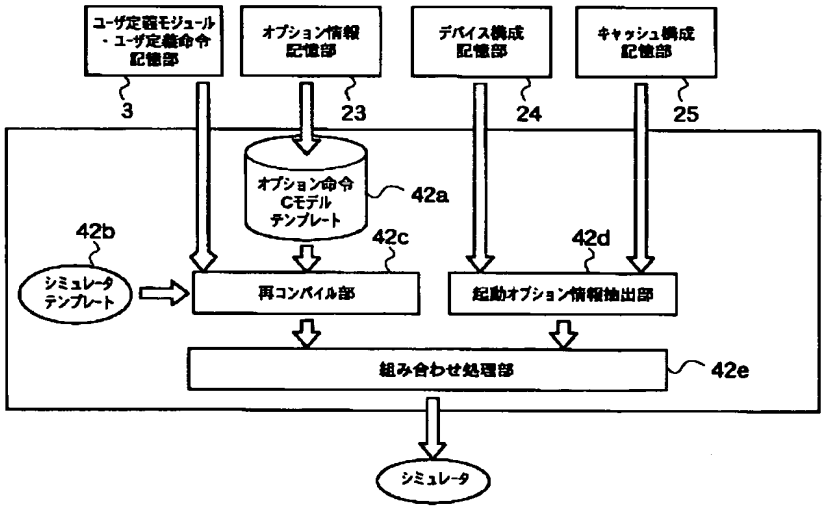
(a)

```
Module_set()
{
   //各コアの設定関数をコールし、オブジェクトの生成と設定を行う
   Core1_config_set() ;
   Core2_config_set() ;
   Core3_config_set() ;
}
```

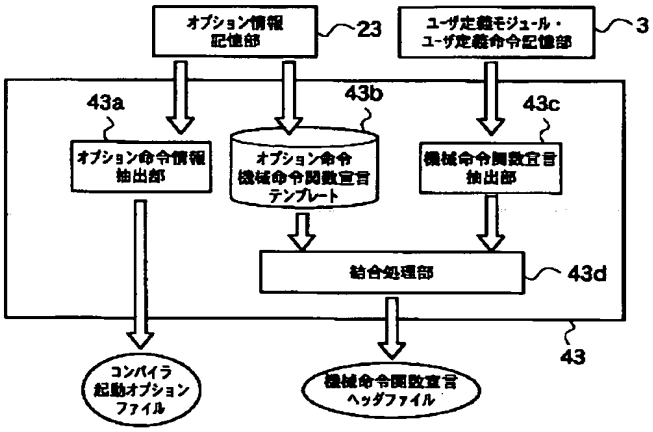
(b)

```
One_set()
{
   Core1>one_step_execute() ;
   Core2>one_step_execute() ;
   Core3>one_step_execute() ;
}
```

【図８】



【図１０】



43：コンパイラカスタマイズ部

【図１１】

(a)

```
-opt_minmax=ON
-opt_div=ON
-opt_bit=OFF
  .
  .
  .
```

(b)

```
XOR      Rn   Rm     ((Rn,32,U,1,1), (Rm,32,U,1,0))
XOR1  Rn, Rm,  Imm8    ((Rn,32,U,1,1), (Rm,32,U,1,0), (Imm8,U,1,0,))

#XOR   Rn,Rmの部分がアセンブラでのフォーマット、
#それに続いてオペランドの定義が続く。

#(Rn,32,U,1,1)でRnはオペランドの表示、32はビット数、
#Uは符号の有無(Uがunsigned、Sがsigned)、次の1は
#Rnがソースオペランドであることを示し、最後の1は
#Rnがデスティネーションオペランドであることを示している。
```

【図１３】

(a)

番号	開始アドレス	終了アドレス	命令数
1	0x8001e	0x80038	370

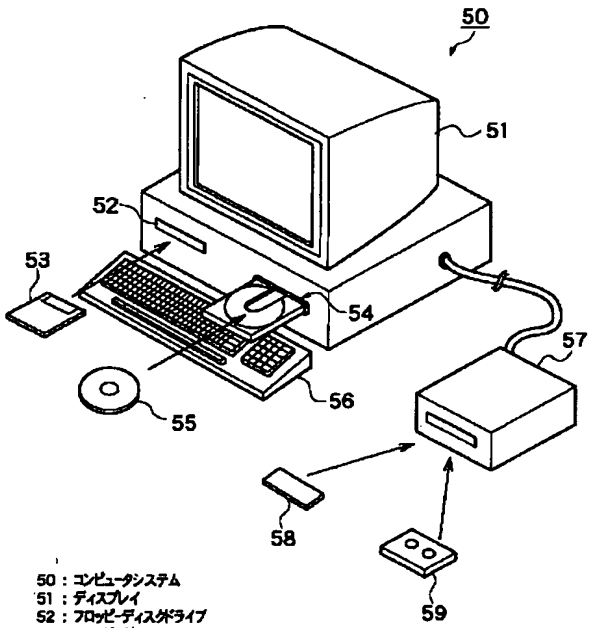
(b)

番号	開始アドレス	終了アドレス	サイクル数
1	0x8001e	0x80038	500

(c)

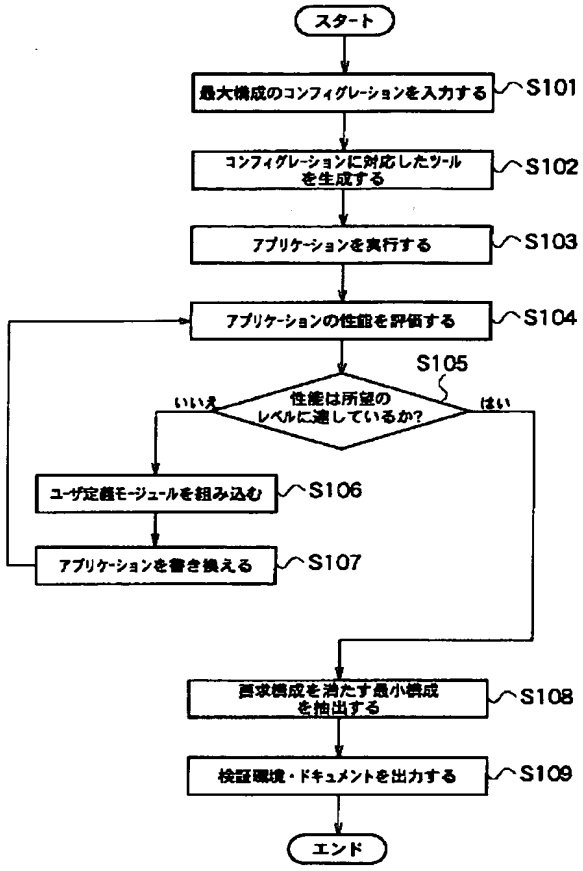
キャッシュサイズ	キャッシュミス率
1 Kbytes	0.191
2Kbytes	0.148
4 Kbytes	0.109
8 Kbytes	0.087
16 Kbytes	0.066

【図１９】



50：コンピュータシステム
51：ディスプレイ
52：フロッピーディスクドライブ
53：フロッピーディスク
54：光ディスクドライブ
55：光ディスク
56：キーボード
57：ドライブ装置
58：ROM
59：カートリッジ

【図１４】



【図１５】

(a)

```
        move R5, 0X00080000
        move R6, 0X00090000
        move R7, 0X00080100
  loop :
        ld          R1,    (R5)
        ld          R2,    4(R5)
        call        _Calc                  //サブルーチンへ
        st          R1,    (R6)
        add         R6,      4
        add         R5,      8
        not_equal       R5,      R7,  loop  //if R5 !=R7 then jump roop
```

(b)

```
Move R1, 0X00080000          // R1に開始アドレスを代入
Move R2, 0X00080100          // R2に終了アドレスを代入
Move R3, 0X00090000          // R3に計算結果収納の開始位置を代入
Move R4, 0X00000001          // R4にDSPスタート用の値を代入
St_cntlbus R1, (cntlbus_addr1)  // READ開始位置を指定
St_cntlbus R2, (cntlbus_addr2)  // READ終了位置を指定
St_cntlbus R3, (cntlbus_addr3)  // 計算結果の書き戻し開始位置を指定
St_cntlbus R4, (cntlbus_addr4)  // DSP処理スタート
(コアプロセッサは、ほかの処理を引き続き行い、DSPの終了を待って処理を継続する)
```

(c)

```
Tyedef unsigned long        address ;
class DSP_HWEngine : public HWEngine、public ControlBus
{
public :
  virtual bool read_cntlbus(address adr, unsigned long* data) ;
  virtual bool write_cntlbus(address adr, unsigned long data) ;
  virtual bool do_command(unsigned long s1,unsigned long s2,unsigned long operand,
                          unsigned long* ret) ;
}
```

【図１６】

```
Bool DSP_HWEngine::read_cntlbus(address adr, unsigned loog* data)
{
        * data=cntlbus[adr];
        return  true;
}
bool DSP_HWEngine::write_cntlbus(address adr, unsigned loog data)
{
        if(adr==cntlbus_addr4)//DSP制御レジスタに書き込みがあった
        {
              書き込んだ値に対して、処理を行う
              return  true;
        }
    cntlbus[adr]=data
    return  true;
}

bool DSP_HWEngine::do_command(unsigned  long s1,unsigned long S2, unsigned long operand,
                                    unsigned long*  ret)
{
if(operand & 0×0000ff00)==0×60)
{
    命令コード0×60の処理を書く。命令へのソースはdo_command関数の引数、s1、s2、
    ディスティネーションは、retである。計算の結果をretにいれて返却する。
    *ret=結果;
    return  true;
    }
ユーザが定義した命令があるときは、if分の数が複数になり、ならぶ。
Else if(……)
{
        ……
    }
}
```

【図１７】

(a)

```
Class CORE
{
public :
    void one_step_execute() ;
};
```

(b)

```
Core1_config_set()
{
        //core1のオブジェクトの生成
        //core1のオプション命令のセット
        //ハードウェアエンジンなどのセット
        //その他の設定(キャッシュ、コプロ)
}

Core2_config_set()
{
        //core2のオブジェクトの生成
        //core2のオプション命令のセット
        //ハードウェアエンジンなどのセット
        //その他の設定(キャッシュ、コプロ)
}

Core3_config_set()
{
        //core3のオブジェクトの生成
        //core3のオプション命令のセット
        //ハードウェアエンジンなどのセット
        //その他の設定(キャッシュ、コプロ)
}
```

フロントページの続き

(51) Int. Cl.[7]	識別記号	Ｆ Ｉ	ﾃｰﾏｺｰﾄﾞ(参考)
Ｇ０６Ｆ 11/26		Ｇ０６Ｆ 11/26	

(72) 発明者 大山 隆一郎
神奈川県川崎市幸区小向東芝町１番地 株式会社東芝マイクロエレクトロニクスセンター内

(72) 発明者 丹後田 愛
神奈川県川崎市幸区小向東芝町１番地 株式会社東芝マイクロエレクトロニクスセンター内

(72) 発明者 内田 勝也
神奈川県川崎市幸区小向東芝町１番地 株式会社東芝マイクロエレクトロニクスセンター内

Ｆターム(参考) 5B046 AA08 BA02 JA04
5B048 AA17 AA20 DD14